A Method for Providing VITAL Model of Embedded Memory with Delay Back Annotation

Background of the Invention

Field of the Invention

[0001] This invention relates generally to electronic design automation systems and methods. More particularly, the present invention relates to design, simulation, and modeling of memory circuits within application specific integrated circuits (ASIC) with delay back annotation using VITAL (VHDL Initiative Towards ASIC Libraries).

Description of Related Art

[0002] Electronic design automation (EDA) systems and methods are presently utilized for the design and verification of electronic integrated circuits, modules containing one or more integrated circuit chips, and printed circuit boards onto which the modules are mounted. Further, the EDA systems and methods are employed in the overall design and verification of electronic systems that contain one or more printed circuit boards.

[0003] EDA systems and methods currently make use of VHDL (VHSIC Hardware Description Language). VHDL is a standard programming language for describing the structure and function of electronic systems that arisen from the United States Government's Very High Speed Integrated Circuits (VHSIC)

program, initiated in 1980. VHDL has been adopted as a standard (IEEE-STD-1076) by the Institute of Electrical and Electronic Engineers (IEEE).

[0004] VHDL allows description of the structure of a design. The structure of the design may then be decomposed into sub-designs. The description then includes the interconnections of the sub-designs. Having the description of the structure of the designs in a familiar programming language form allows the design to be simulated before being manufactured. The simulation permits the designers to quickly compare alternatives and test for correctness without the delay and expense of hardware prototyping.

[0005] "A Tutorial Introduction to VITAL", Steven E. Schulz, P.E., Presented at the 1995 Mentor Users' Group Conference, October 24, 1995 provides a description of the proposed VITAL specification and is summarized in part hereinafter.

[0006] During the early 1990's, as ASIC gate densities increased dramatically, there were higher levels of abstraction in the design of electronic systems. Those companies creating EDA methods and systems were making rapid advances in circuit synthesis. With increasing improvements in computing systems, communication systems, and other electronic applications, there were increased pressure to reduce the design cycle time. VHDL was not able to advance with the requirements because of a lack of an ASIC design methodology, a simulation performance bottleneck, insufficient timing accuracy for verification of design before manufacture, and there were no standard ASIC

circuit and sub-circuit libraries. To ameliorate these problems, the VHDL Initiative Towards ASIC Libraries (VITAL) was formed. The VITAL specification was adopted by the IEEE as IEEE STD-1076.4 in 1995 and is incorporated herein by reference.

[0007] The purpose of VITAL was to accelerate the availability of ASIC libraries across EDA manufactureres of VHDL simulators. The priorities of the VITAL ASIC libraries were to provide (1) high accuracy for sub-micron ASICs, (2) fast simulation performance; and (3) an aggressive schedule. The VITAL specification consists of standardized:

Timing Routines

Primitive models

Instance Delay Loading Mechanism

Model Development Guidelines Document.

[0008] The VITAL specification provided ASIC designers improved portability of designs and libraries across EDA tools, fast simulation performance without leaving VHDL design environment, and the ability to use standard VITAL routines in user-written models. The advantages for integrated circuit manufactureres are that a single library supports all major EDA simulation environments; there is high-accuracy timing support for deep sub-micron circuit designs, and consequently reduced development and maintenance costs. The standard libraries of the VITAL specification allow EDA Vendors to focus on tool and

design issues rather than libraries. The standard modeling techniques allow improved optimization and reduced complexity.

[0009] The VITAL specification allows flexible specification of functionality with table lookup primitives (w/ multiple outputs), Boolean primitives (specific number of inputs or programmable number of inputs), and behavioral or concurrent style. Further, the VITAL specification allows accurate specification of timing delays. The timing delays may be pin-to-pin or distributed, state-dependent, or conditional. Accurate timing check support is created by having setup and hold timing verification (including negative constraints), recovery/removal checks, minimum pulsewidth, period checks, glitch on event, glitch on detect, and no glitch elements.

[0010] The VITAL specification of 1995 does not provide a method of modeling a memory, particularly, a static random access memory (SRAM) or a Flash non volatile random access memory (Flash NVRAM). "The Standard VITAL ASIC Modeling Specification" (VITAL 2000) IEEE 1076.4, Feb. 2001, provides support for all types of SRAM and ROM memories - single, dual port, and multi-port architectures with synchronous and asynchronous operation. The VITAL 2000 specification provides support for:

bit/sub-word/word addressability.

timing and functional violations.

corruption handling, contention policies.

address validation for out of range, or invalid value.

negative timing constraints.

memory initialization from an external file.

standard delay format (SDF) back annotation.

[0011] "VHDL Sign-Off Simulation: What Future?," Bakowski, et al., Proceedings of the VHDL International Users Forum. Spring Conference, 1994, IEEE, May 1994, pp. 136 - 141, describes that a universal VHDL gate library for ASIC sign-off simulation can be developed, even though the optimized VHDL code for various target simulators may differ. The solution is based on VHDL models written with a unique entity declaration and various architecture bodies targeted at simulators, concentrating on VITAL compliant architectures.

[0012] " Standardizing ASIC Libraries in VHDL Using VITAL: a Tutorial," Krolikoski, Proceedings of the IEEE 1995 Custom Integrated Circuits Conference, IEEE, May 1995, pp. 603 - 610, provides an overview of the central features of VITAL. Models using several VITAL-compliant styles are presented and discussed.

[0013] "IEEE Standard for VITAL Application Specific Integrated Circuit (ASIC) Modeling Specification," Design Automation Standards Committee of the IEEE Computer Society, USA, IEEE Std 1076.4-1995, May 1996, defines the VITAL (VHDL Initiative Towards ASIC Libraries) ASIC Modeling Specification.

[0014] U. S. Patent 6,026,226 (Heile, et al.) describes a technique for allowing local compilation at any level within a design hierarchy tree for a programmable

logic device. The technique allows a user to compile an entire design using inherited parameter values and assignments from any parent nodes within the design hierarchy tree.

[0015] U. S. Patent 5,875,111 (Patel) describes a method of modeling a pull-up device and a pull-down device with delay back annotation in accordance with the VITAL application specific integrated circuit modeling specification.

[0016] U. S. Patent 6,141,631 (Blinne, et al.) describes a method for determining the behavior of a VHDL model of a logic cell, which receives input signals resulting in a narrow pulse or "glitch." If the pulse width of the output pulse is narrower than a pulse rejection period, the output pulse is rejected and is not propagated to subsequent logic cells connected to the output.

Summary of the Invention

[0017] An object of this invention is to provide a VITAL model for a memory such as an SRAM or a flash NVRAM.

[0018] Another object of this invention is to provide a VITAL model of this invention where VITAL path delay procedures are overloaded for timing of address, control, and data bus signals to the memory.

[0019] To accomplish at least one of these objects, a method for modeling a memory with delay back annotation in accordance with the VITAL application specific integrated circuit modeling specification begins with modeling the

memory with a timing generic and a port declarations. The wire delay of the memory is then modeled, followed by modeling a timing check for the memory. The wire delay of the model of the memory is then created. A description of the functional operation of the memory is then generated. The path delay for the address, control, and data bus signals to the memory is formed by overloading the VITAL path delay procedures. The VITAL timing check procedures are overloaded to determine the timing constraint violations of the timing bus signals of the memory. The VITAL wire delay procedures are overloaded to determine interconnection delay bus signals of the memory.

Brief Description of the Drawings

[0020] Fig. 1 is a block diagram of an electronic design automation system incorporating a VITAL Model of Embedded Memory with Delay Back Annotation of this invention.

[0021] Fig. 2 is a flowchart of a method for designing a VITAL Model of Embedded Memory with Delay Back Annotation of this invention.

[0022] Figs. 3-5 are a flowcharts illustrating the modeling of a path delay of a memory by overloading VITAL path delay procedures.

Detailed Description of the Invention

[0023] The design of electronic circuits and systems encompass three fundamental domains: the functional domain, the structural domain, and the

geometric domain. The functional domain describes the algorithms performed by the system as expressed in a register transfer language, Boolean equations, and differential equations. The modeling of the functional domain is commonly referred to as behavioral modeling, which describes the functional operation of the system.

[0024] The structural domain higher level circuit function that are then further detailed as register, combinatorial, and operational functions, These functions are then detailed as circuits composed of the various electronic components. The higher level circuit functions are described as higher level abstractions within VHDL or as functional circuit blocks within a schematic capture program.

[0025] The geometric domain describes the actual floorplan of a system on chip, module, or printed circuit board. The floorplan is then decomposed into the individual components or cells of the system being constructed. The components and the interconnecting wiring are formed to create appropriate levels of polygons representing the description of the levels of materials required to form the electronic components and the interconnecting wiring.

[0026] To create the design of electronic circuits and systems, more of the operations and documentation of these designs are performed by computer systems executing programs for defining the functional structure and geometric domains for the circuits and system. An example of an electronic design automation computer system is shown in Fig. 1. The EDA system has a central processor **10** which includes an execution unit **15** and a main memory **20**. The

central processor retrieves electronic design automation programs from the central program retention device **25**, and places the compiled programs in the main memory **20**. The electronic design automation programs are executed by using the descriptions of the electronic circuits from the model description retention device **30**. The model description calls standardized component and function descriptions from a model library contained on the library retention device.

[0027] The model descriptions begin with functional descriptions that are often in a register transfer language. The functional descriptions are then parsed and validated for correctness in order to determine whether the functional description does describe the desired operation. When the functional description is completed, a compiler is often executed on the execution unit **15** to begin the detailing of the electronic design in the structural domain and to create logical and circuit descriptions that detail the electronic components necessary to complete the design. From the structural domain, the electronic components are allocated by physical design programs executed on the execution unit **15** to specific areas of the topography of the physical chip, module, or printed circuit being designed. The EDA physical design programs develop the necessary physical entities that are to be used to manufacture the electronic components within the overall design. Further, as the physical description of the geometric domain is created, the electrical characteristics, such as power dissipation, current requirements, time delays of the circuits and components are determined and referred back or back annotated to the structural domain and the functional

domain for further simulation of the functional performance to verify the correctness of the design.

[0028] Each program provides a domain for the design that causes the execution unit **15** to essentially function as an independent machine that performs the appropriate functions to create the various parts of the functional, structural, and geometric domains. The processor **10** is connected to workstations **40a**, **40b**, ..., **40n** such that the designer of the electronic circuit or system can monitor and provide necessary information and modification during the progress of the design. Further, the processor **10** may be connected to a network **45** such that either other EDA computing systems or other workstations may be allowed access to the EDA programs, the model libraries, or the electronic circuit designs.

[0029] Refer now to Fig. 2 for a description of the flow of the program processes as executed on the processor **10** of Fig. 1. As described above, VHDL is a hardware description language that is employed to assist the design of electronic circuits and systems. The syntax and format of VHDL, as defined in the IEEE Std. 1076, allows a designer to describe the functional entities (**Box 100**) of the electronic circuit or system being designed. The VITAL standard library (**Box 120**) provides the standard structure of generalized electronic components provided particularly by ASIC manufacturers. The VITAL standard library (**Box 120**) is used to assist the designer to create the necessary entities for the electronic design. Once the functional and structural entities (**Box 100**) are

created, the basic architecture (**Box 105**) of the physical geometric design is created. Again, the VITAL standard library (**Box 120**) contains the fundamental geometric description for each of the entities of the design as provided by the creator of the library (ASIC manufacturer). From the geometric descriptions of the interconnections of the design, an estimate of the interconnection wiring lengths is created and from the wiring lengths, the delay resulting from the circuit interconnections or wire delay is calculated (**Box 110**). The modeled circuit delays from the VITAL standard library (**Box 120**) and the wire delays are collected to generate a timing check (**Box 115**) to validate that the circuit meets the necessary timing objectives.

[0030] Generics within the entities of models that comply with the VITAL specification provide specific kinds of timing and control information. In the preferred embodiment of this invention for an SRAM or NVRAM VITAL description, the timing generics created (**Box 100**) within the entities for the SRAM or NVRAM are described as follows:

```
tpd_XE_DOUT          : VitalDelayArrayType01Z(numOut-1 downto
      0):=(others =>(Txa, Txa, Txa Txa, Txa, Txa));
tpd_YE_DOUT          : VitalDelayArrayType01Z(numOut-1 downto 0)
      :=(others =>(Tya, Tya, Tya, Tya, Tya, Tya));
tpd_OE_DOUT          : VitalDelayArrayType01Z(numOut-1 downto 0)
      :=(others =>(Toa, Toa, Toa, Toa, Toa, Toa));
tpd_SE_DOUT          : VitalDelayArrayType01Z(numOut-1 downto 0)
      :=(others =>(Tya, Tya, Tya, Tya, Tya. Tya));
```

[0031] The a timing check (**Box 115**) functions such as VITALSetupHoldCheck, VITALRecoveryRemovalCheck, and VITALPeriodPulseCheck maybe modeled as follows:

```
for i in numAddrX-1 downto 0 loop
   VitalSetupHoldCheck
   Violation          =>Tvol_XADR_ERASE_1,
   TimingData`        =>TimingData_XADR_ERASE 1,
   TestSignal               =>XADR_ipd,
   TestSignalName=> "XADR",
   TestDelay                => 0 ns,
   RefSignal          => ERASE_ipd,
   RefSignalName  => "ERASE"
   RefDelay           => 0 ns,
   HoldHigh           => Tnvh1,
   CheckEnabled   => (To_X01Z(MAS1_ipd) = '1'),
   RefTransition      => 'F',
   HeaderMsg          =>InstancePath & "/SFB0008_08B9_CORE_INFO",
   Xon                => Xon
   MsgOn              => MsgOn,
   MsgSeverity        => WARNING);
end loop;
```

[0032] Once the timing of the interconnected circuits is verified (**Box 115**), the total functionality (**Box 125**) of the design is then simulated. Often this involves simulation of individual circuits, the results of which are then passed to a global function simulator such that the entire circuit functionality can be proven.

[0033] Once the functioning of the design is verified and the actual geometric design for all the electronic components and the interconnections are completed, the critical paths of the design are then calculated (**Box 130**). In the VITAL specification, the path delay section provides the procedures that drive ports or internal signals using appropriate delay values. The procedures have provisions for glitch handling, message reporting control, and output strength mapping. Path delay selection within the VITAL specification is modeled with a procedure call statement that invokes one of the path delay procedures —VITALPathDelay, VITALPathDelay01, or VITALPathDelay01Z — defined in the package

VITAL_Timing. A path delay procedure selects the appropriate propagation delay path and schedules a new output value for the specified signal.

[0034] To assist in thoroughly verifying the functioning of the electron circuit or system, the path delay of the actual physical design must be determined and fed back or back annotated to the model of the circuit such that the simulations for the circuit or system are more accurate. As stated above, the VITAL specification of 1995 does not provide a method of modeling a memory, in particular a static random access memory (SRAM) or a Flash non volatile random access memory (Flash NVRAM). To overcome this problem, this invention provides a method for overloading the path delay procedures of the VITAL specification. An SRAM or NVRAM is described according VITAL specification as shown in the APPENDIX. The path delay procedures are overloaded (**Box 135**) in order to generate the path delay timings for the input address, data, and control buses and output buses of the SRAM or NVRAM.

[0035] Overloading, as is known in the art, allows two procedures written in VHDL to have the same name, provided the number or base types of these parameters differs. When a call to an overloaded procedure is made the number of actual parameters, their order, their base types and the corresponding formal parameter names (if named association is used) are used to determine which subprogram is meant. This permits the standard path delay procedures to be expanded to accommodate the SRAM and NVRAM structure for embedding these structures within an ASIC.

[0036] Refer now to Figs. 3 - 5 for an explanation of the overloading of the Vital path delay procedures. The predefined path delay procedures VitalPathDelay, VitalPathDelay01, and VitalPathDelay01Z, each provide the following capabilities:

- Transition dependent path delay selection.
- User controlled glitch detection, "Don't Care ('X') generation, and violation reporting.
- Scheduling of the computed values on the specified signal.

Selection of the appropriate path delay begins with the selection of candidate paths. The candidate paths are selected by identifying the paths for which the Path Condition is true. If there is a single candidate path then its delay is the one selected. If there is more than one candidate path, then the shortest delay (accounting for the InputChangeTime parameter) is selected using transition dependent delay selection. If there are no candidate paths then the delay specified by the DefaultDelay parameter to the path delay procedure is used.

[0037] The VitalPathDelay and VitalPathDelay01 procedures schedule path delays on signals for which the transition to High Impedance ('Z') is not important. These procedures are distinguished from one another by the type of delay values that they accept. The procedure VitalPathDelay is defined for simple path delays of type VitalDelayType. Procedure VitalPathDelay01 is defined for transition dependent path delays of type VitalDelayType01 (rise/fall delays). The Procedure VitalPathDelay01Z schedules path delays on signals for

which the transition to or from High Impedance ('Z') is important (e.g., modeling of tri-state drivers). In addition to the basic capabilities provided by all path delay procedures, VitalPathDelay01Z performs result mapping of the output value (using the value specified by the actual associated with the OutputMap parameter) before scheduling this value on the signal. This result mapping is performed after a transition dependent delay selection but before scheduling the final output.

[0038] In Fig. 3 the VITAL Path Delay procedure is overloaded by first declaring (**Box 200**) a new MemoryVITALPathTYPE as follows:

```
Type MEMORYVitalPathType is record
InputChangeTime : time;          -- timestamp for path input signal
PathDelay : VitalDelayArrayType(numOut -1 downto 0);--Delay for this
path
PathConditon : Boolean;          --path sensitize condition
End record;
```

[0039] The MemoryVITALPathArrayType is then declared (**Box 205**) as follows:

```
Type MEMORYVitalPath ArrayType is array (natural range <> ) of
MEMORYVitalPathType;
```

[0040] The new source code for the new VITAL path delay procedures for determining the path delays for the SRAM and NVRAM is then created (**Box 210**) according to the following:

```
Procedure VitalPathDelay (
  Signal     OutSignal      : out std_logic_vector;
```

```
Variable   GlitchData      : inout VitalGlitchDataArrayType;
Constant   OutSignalName: in    string;
Constant   OutTemp             : in    std_logic_vector;
constant   Paths          : in    MEMORYVitalPathArrayType;
constant   DefaultDelay   : in    VitalDelayType   := VitalZeroDelay;
constant   Mode           : in    VitalGlitchKindType := OnEvent;
constant   XOn                  : in    Boolean          := true;
constant   MsgOn          : in    Boolean := true;
constant   MsgSeverity    : in    SEVERITY_LEVEL :=WARNING
)
```

[0041] At this same time the NewVITALGlitch procedure is created (**Box 215**) according to the following:

```
Procedure Vital Glitch (
  signal OutSignal          : in std_logic_vector;
  variable GlitchData       : inout VitalGlitchDataArrayType;
  constant OutSignalName    : in  string;
  constant New Value        : in  std_logic_vector;
  constant NewDelayArray    : in  PropDelayArrayType;
  constant Mode                   : in   VitalGlitchKindType := On Event;
  constant XOn                    : in   Boolean             := true;
  constant MsgOn                  : in Boolean               := FALSE;
  constant MsgSeverity            : in  SEVERITY_LEVEL := WARNING
)
```

[0042] The original source code for the VITALPathDelay procedure as described in the specification is created (**Box 220**) and merged with the new source code (**Box 210**) and the NewVITALGlitch procedure (**Box 215**) to form (**Box 225**) the overloaded VITALPathDelay procedure

[0043] In Fig. 4 the VITALPathDelay01 procedure is overloaded by first declaring (**Box 230**) a new MemoryVITALPathType01 as follows:

```
Type MEMORYVITALPathType01 is record
InputChangeTime : time;          -- timestamp for path input signal
PathDelay : VitalDelayArrayType(numOut -1 downto 0);--Delay for this
path
PathConditon : Boolean;          --path sensitize condition
End record;
```

[0044] The MemoryVITALPathArrayType is then declared (**Box 235**) as follows:

```
Type MEMORYVitalPath ArrayType is array (natural range <> ) of
MEMORYVITALPathType01;
```

[0045] The new source code for the new VITAL path delay procedures for determining the path delays for the SRAM and NVRAM is then created (**Box 240**) according to the following:

```
Procedure VitalPathDelay01 (
  Signal    OutSignal     : out std_logic_vector;
  Variable  GlitchData    : inout VitalGlitchDataArrayType;
  Constant  OutSignalName : in  string;
  Constant  OutTemp       : in  std_logic_vector;
  constant  Paths         : in  MEMORYVitalPathArrayType;
  constant  DefaultDelay  : in  VitalDelayType  := VitalZeroDelay;
  constant  Mode          : in  VitalGlitchKindType := OnEvent;
  constant  XOn           : in  Boolean         := true;
  constant  MsgOn         : in  Boolean := true;
  constant  MsgSeverity   : in  SEVERITY_LEVEL :=WARNING
)
```

[0046] At this same time the NewVITALGlitch procedure is created (**Box 245**) according to the following:

```
Procedure VitalGlitch (
  signal OutSignal          : in std_logic_vector;
  variable GlitchData       : inout VitalGlitchDataArrayType;
  constant OutSignalName    : in  string;
  constant New Value        : in  std_logic_vector;
  constant NewDelayArray    : in  PropDelayArrayType;
  constant Mode                   : in  VitalGlitchKindType := On Event;
  constant XOn                    : in  Boolean             := true;
  constant MsgOn                  : in Boolean              := FALSE;
  constant MsgSeverity            : in  SEVERITY_LEVEL := WARNING
)
```

[0047] The original source code for the VITALPathDelay01 procedure as described in the specification is created (**Box 250**) and merged with the new source code (**Box 240**) and the NewVITALGlitch procedure (**Box 245**) to form (**Box 255**) the overloaded VITALPathDelay01 procedure

[0048] In Fig. 5 the VITALPathDelay01Z procedure is overloaded by first declaring (**Box 260**) a new MemoryVITALPathType01Z as follows:

```
Type MEMORYVITALPathType01Z is record
InputChangeTime : time;          -- timestamp for path input signal
PathDelay : VitalDelayArrayType(numOut -1 downto 0);--Delay for this
path
PathConditon : Boolean;          --path sensitize condition
End record;
```

[0049] The MemoryVITALPathArrayType is then declared (**Box 265**) as follows:

```
Type MEMORYVitalPath ArrayType  is array (natural range <> ) of
MEMORYVITALPathType01Z;
```

[0050] The new source code for the new VITAL path delay procedures for determining the path delays for the SRAM and NVRAM is then created (**Box 270**) according to the following:

```
Procedure VitalPathDelay01Z (
  OutSignal              => DOUT
  GlitchData             => DOUT_GlitchData,
  OutSignal Name         => "DOUT",
  OutTemp                => DOUT_zd,
  Paths                  => (0 => (XE_ipd'last_event, ZeroDelay, true),
                             1 => (YE_ipd'last_event, ZeroDelay, true),
                             2 => (OE_ipd'last_event, ZeroDelay, true),
                             3 => (SE_ipd'last_event, ZeroDelay, true),
                             4 => (XADR_ipd'last_event, ZeroDelay, true),
                             5 => (YADR_ipd'last_event, ZeroDelay, true),
  Mode                   =>On Detect,
  XON                    =>XON
  MsgOn                  =MsgOn,
  MsgSeverity            =>WARNING);
```

[0051] At this same time the NewVITALGlitch procedure is created (**Box 275**) according to the following:

```
Procedure VitalGlitch (
  signal OutSignal          : in std_logic_vector;
  variable GlitchData       : inout VitalGlitchDataArrayType;
  constant OutSignalName    : in  string;
  constant New Value        : in  std_logic_vector;
  constant NewDelayArray    : in  PropDelayArrayType;
```

```
        constant Mode              : in  VitalGlitchKindType := On Event;
        constant XOn               : in  Boolean             := true;
        constant MsgOn             : in Boolean              := FALSE;
        constant MsgSeverity       : in  SEVERITY_LEVEL := WARNING
    )
```

[0052] The original source code for the VITALPathDelay01Z procedure as described in the specification is created (**Box 280**) and merged with the new source code (**Box 270**) and the NewVITALGlitch procedure (**Box 275**) to form (**Box 285**) the overloaded VITALPathDelay01Z procedure

[0053] While this invention has been particularly shown and described with reference to the preferred embodiments thereof, it will be understood by those skilled in the art that various changes in form and details may be made without departing from the spirit and scope of the invention.

APPENDIX

```
--------------------------------
-- Copyright (c) 1997 by Taiwan Semiconductor Manufacturing Company Ltd.
-- All Rights Reserved. No part of this publication may be reproduced in
-- whole or part by any means without the prior written consent.
-------------------------------------------------------------------------
-
-- LICENSE:  Taiwan Semiconductor Manufacturing Company ("TSMC") grants you
-- the non-exclusive right to use the enclosed software
-- program.  You will not use, copy, modify,
-- display, rent, sell or transfer the Software or any portion
-- thereof, except as provided in this Agreement.
-------------------------------------------------------------------------
-
-- Title:           TSMC SFB0008_08B9_CORE_MAIN VITAL Model
-- File Name:       SFB0008_08B9_core_main.vhd
-- File Contents:   package, entity, architecture and configuration
-- Entity:          SFB0008_08B9_CORE_MAIN
-- Architecture:    BEHAVIORAL_SFB0008_08B9_CORE_MAIN_VITAL
-- Used Subdesign:
--                  SFB0008_08B9_spec_main.vhd
--                  SFB0008_08B9_spec_timing.vhd
```

```
--                  tsmc_utility.vhd
--                  SFB0008_08B9_hotpatch_main.vhd
-- Time Scale:      1 ns
-- Logic System:    IEEE-1164 VITAL-95
-- Created By:      Nai-Yin Sung
-- Created Date:    09/09/1998
-- Version:         2.0
--------------------------------------------------------------------------------
-
-- Description:
--------------------------------------------------------------------------------
-
--------------------------------------------------------------------------------
-
-- Note:
--
--
--------------------------------------------------------------------------------
-
-- Modified History:
-- TO BE DONE:
--    functional modeling    11/14/97'
--    timing modeling        11/18/97'
--    verification           11/20/97'
--      cumulative check
--      path delay
--      setup/hold check
--    attached specifically  12/11/97'
--    UNKNOWN condition and
--    delay racing in VHDL
--    (1)
--                  ------------------------
--                  |<-    delay         ->|
--                  |                      |
--    XE    -------                        -----------------
--
--
--
--                       -------------
--                       |<- delay ->|
--                       |           |
--    OE    -------------             ------------------------
--
--
--   version 1.0
--
--   During VITAL or Verilog Simulator event-driven method,
--   therefore, simulator will select the LAST signal event
--   to pass delay. The result will have a period of mistake
--   in simulation results
--
--
--         ------------------------------------------------
--    DOUT       High Impedance     |   Output
--         ------------------------------------------------
--
```

```
--    version 2.0
--
--    Solving this problem, model will reflect reality of
--    circuit.
--
--
--         ------------------------------------------------
--     DOUT       High Impedance    | "X" | Output
--         ------------------------------------------------
--
--     (2) XADR
--
--         ------------------------------------------------
--     XADR  "0" |     "1"          |     "2"
--         ------------------------------------------------
--
--     version 1.0
--
--              |<- delay ->|     |<- delay ->|
--         ------------------------------------------------
--     DOUT      "0"          |  "1"             | "2"
--         ------------------------------------------------
--
--     version 2.0
--
--              |<- delay ->|     |<- delay ->|
--         ------------------------------------------------
--     DOUT "0"  |    "X"    |  "1"|    "X"    | "2"
--         ------------------------------------------------
-------------------------------------------------------------------------------
-
--     Ports
--         XADR : X Address input buffer
--         YADR : X Address input buffer
--         DIN  : data input bus
--         DOUT : data output bus
--         XE   : X address enable
--         YE   : Y address enable
--         SE   : sense amplifier enable
--         OE   : output enable
--         ERASE: erase cycle
--         MAS1 : mass erase cycle
--         PROG : program cycle
--         NVSTR: non-volatile store cycle
--         CE   : chip select
-------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_textio.all;
use std.textio.all;
use ieee.vital_primitives.all;
use ieee.vital_timing.all;
use work.SFB0008_08B9_spec_main.all;
use work.SFB0008_08B9_spec_timing.all;
use work.tsmc_utility.all;
```

```
use work.SFB0008_08B9_hotpatch_main.all;
entity SFB0008_08B9_CORE_MAIN is
      generic (
            TimingCheckOn                          : Boolean := true;
            InstancePath                           : String := "*";
            Xon                                    : Boolean := True;
            MsgOn                                  : Boolean := True;
            DebugMode                              : Boolean := True; -- NISUNG
            tpd_XE_DOUT                            :
VitalDelayArrayType01Z(numOut-1 downto 0) := (others =>(Txa, Txa, Txa, Txa,
Txa, Txa));
            tpd_YE_DOUT                            :
VitalDelayArrayType01Z(numOut-1 downto 0) := (others =>(Tya, Tya, Tya, Tya,
Tya, Tya));
            tpd_OE_DOUT                            :
VitalDelayArrayType01Z(numOut-1 downto 0) := (others =>(Toa, Toa, Toa, Toa,
Toa, Toa));
            tpd_SE_DOUT                            :
VitalDelayArrayType01Z(numOut-1 downto 0) := (others =>(Tya, Tya, Tya, Tya,
Tya, Tya));
--          tpd_XADR_DOUT                          :
VitalDelayArrayType01Z(numOut-1 downto 0) := (others =>(Toa, Toa, Toa, Toa,
Toa, Toa));
--          tpd_YADR_DOUT                          :
VitalDelayArrayType01Z(numOut-1 downto 0) := (others =>(Toa, Toa, Toa, Toa,
Toa, Toa));
            tsetup_DIN_YE_noedge_posedge  : VitalDelayArrayType(numOut-1
downto 0) := (others => Tads);
            thold_DIN_YE_noedge_negedge          : VitalDelayArrayType(numOut-
1 downto 0) := (others => Tadh);
            tsetup_ERASE_NVSTR_posedge_posedge  : VitalDelayType := Tnvs;
            tsetup_XE_NVSTR_posedge_posedge           : VitalDelayType :=
Tnvs;
            tsetup_XADR_NVSTR_noedge_posedge     :
VitalDelayArrayType(numAddrX-1 downto 0) := (others => Tnvs);
            thold_NVSTR_ERASE_negedge_negedge    : VitalDelayType := Tnvh; --
NISUNG
            thold_XE_ERASE_negedge_negedge            : VitalDelayType :=
Tnvh; -- NISUNG
            thold_XADR_ERASE_negedge_negedge     :
VitalDelayArrayType(numAddrX-1 downto 0) := (others => Tnvh); -- NISUNG
            thold_MAS1_ERASE_negedge_negedge     : VitalDelayType := Tnvh1; --
NISUNG
            tsetup_MAS1_NVSTR_posedge_posedge    : VitalDelayType := Tnvs;
            tsetup_PROG_NVSTR_posedge_posedge    : VitalDelayType := Tnvs;
            thold_XE_PROG_negedge_negedge        : VitalDelayType := Tnvh;
            thold_NVSTR_PROG_negedge_negedge     : VitalDelayType := Tnvh;
            thold_XADR_PROG_negedge_negedge           :
VitalDelayArrayType(numAddrX-1 downto 0) := (others => Tnvh);
            tsetup_NVSTR_YE_posedge_posedge           : VitalDelayType :=
Tpgs;
            tsetup_YADR_YE_noedge_posedge        :
VitalDelayArrayType(numAddrY-1 downto 0) := (others => Tads);
            thold_YADR_YE_noedge_negedge         :
VitalDelayArrayType(numAddrY-1 downto 0) := (others => Tadh);
            thold_PROG_YE_negedge_negedge        : VitalDelayType := Tpgh;
```

```
            trecovery_NVSTR_MAS1_negedge_posedge      : VitalDelayType :=
Trcv;
            trecovery_NVSTR_XE_negedge_posedge        : VitalDelayType :=
Trcv;
            trecovery_NVSTR_ERASE_negedge_posedge     : VitalDelayType :=
Trcv;
            trecovery_NVSTR_PROG_negedge_posedge      : VitalDelayType :=
Trcv;
            trecovery_NVSTR_YE_negedge_posedge        : VitalDelayType :=
Trcv;
            tpw_NVSTR_posedge                         : VitalDelayType :=
Terase;
            tpw_YE_posedge                            : VitalDelayType :=
Tprog;
            tipd_XADR                                 :
VitalDelayArrayType(numAddrX-1 downto 0) := (others => (0.0 ns));
            tipd_YADR                                 :
VitalDelayArrayType(numAddrY-1 downto 0) := (others => (0.0 ns));
            tipd_DIN                                  :
VitalDelayArrayType(numOut-1 downto 0) := (others => (0.0 ns));
            tipd_XE                                         :
VitalDelayType01 := (0.0 ns, 0.0 ns);
            tipd_YE                                         :
VitalDelayType01 := (0.0 ns, 0.0 ns);
            tipd_SE                                         :
VitalDelayType01 := (0.0 ns, 0.0 ns);
            tipd_OE                                         :
VitalDelayType01 := (0.0 ns, 0.0 ns);
            tipd_ERASE                                : VitalDelayType01 :=
(0.0 ns, 0.0 ns);
            tipd_MAS1                                 : VitalDelayType01 :=
(0.0 ns, 0.0 ns);
            tipd_PROG                                 : VitalDelayType01 :=
(0.0 ns, 0.0 ns);
            tipd_CE                                         :
VitalDelayType01 := (0.0 ns, 0.0 ns);
            tipd_NVSTR                                : VitalDelayType01 :=
(0.0 ns, 0.0 ns)
      );
      port(
            XADR              : in std_logic_vector(numAddrX-1 downto 0);
            YADR              : in std_logic_vector(numAddrY-1 downto 0);
            DIN               : in std_logic_vector(numOut-1 downto 0);
            XE                      : in std_logic;
            YE                      : in std_logic;
            SE                      : in std_logic;
            OE                      : in std_logic;
            ERASE             : in std_logic;
            MAS1              : in std_logic;
            PROG              : in std_logic;
            NVSTR             : in std_logic;
            CE                      : in std_logic;
            TMR               : in std_logic;
            DOUT              : out std_logic_vector(numOut-1 downto 0)
      );
      attribute VITAL_LEVEL0 of SFB0008_08B9_CORE_MAIN : entity is TRUE;
```

```
end SFB0008_08B9_CORE_MAIN;
architecture BEHAVIORAL_SFB0008_08B9_CORE_MAIN_VITAL of
SFB0008_08B9_CORE_MAIN is
      attribute VITAL_LEVEL0 of BEHAVIORAL_SFB0008_08B9_CORE_MAIN_VITAL :
architecture is TRUE;
-------------------------------------------------------------
--    Signal connected from wire to SFB0008_08B9_CORE_MAIN
-------------------------------------------------------------
      signal XADR_ipd   : std_logic_vector(numAddrX-1 downto 0) := (others =>
'X');
      signal YADR_ipd   : std_logic_vector(numAddrY-1 downto 0) := (others =>
'X');
      signal DIN_ipd    : std_logic_vector(numOut-1 downto 0) := (others =>
'X');
      signal XE_ipd     : std_logic := 'X';
      signal YE_ipd     : std_logic := 'X';
      signal SE_ipd     : std_logic := 'X';
      signal OE_ipd     : std_logic := 'X';
      signal ERASE_ipd  : std_logic := 'X';
      signal MAS1_ipd        : std_logic := 'X';
      signal PROG_ipd        : std_logic := 'X';
      signal CE_ipd          : std_logic := 'X';
      signal NVSTR_ipd  : std_logic := 'X';

-------------------------------------------------------------
--    Other Signal
-------------------------------------------------------------
      signal EN               : std_logic := '0';  -- Output Enable
      signal Prog_Enable      : std_logic := '0';  -- Program Enable
      signal Erase_Enable     : std_logic := '0';  -- Erase Enable
      signal Output_X_XE      : std_logic := '1';  -- Timing Check for XE
      signal Output_X_XADR    : std_logic := '1';  -- Timing Check for XADR
      signal Output_X_YE      : std_logic := '1';  -- Timing Check for YE
      signal Output_X_YADR    : std_logic := '1';  -- Timing Check for YADR
      signal Output_X_SE      : std_logic := '1';  -- Timing Check for SE
      signal Output_X_OE      : std_logic := '1';  -- Timing Check for OE
      signal Address          : std_logic_vector(numAddrX+numAddrY-1 downto
0) := (others => 'X');

      signal P3Violation                : std_logic := '0';  -- NISUNG
begin
-------------------------------------------------------------
-- Wire Delay Section
-------------------------------------------------------------
WireDelay : block
begin
      L1 : for i in 0 to numAddrX-1 generate
            VitalWireDelay(XADR_ipd(i), XADR(i), tipd_XADR(i));
            end generate;
      L2 : for i in 0 to numAddrY-1 generate
            VitalWireDelay(YADR_ipd(i), YADR(i), tipd_YADR(i));
            end generate;
      L3 : for i in 0 to numOut-1 generate
            VitalWireDelay(DIN_ipd(i), DIN(i), tipd_DIN(i));
            end generate;
      VitalWireDelay(XE_ipd, XE, tipd_XE);
```

```
        VitalWireDelay(YE_ipd, YE, tipd_YE);
        VitalWireDelay(SE_ipd, SE, tipd_SE);
        VitalWireDelay(OE_ipd, OE, tipd_OE);
        VitalWireDelay(ERASE_ipd, ERASE, tipd_ERASE);
        VitalWireDelay(MAS1_ipd, MAS1, tipd_MAS1);
        VitalWireDelay(PROG_ipd, PROG, tipd_PROG);
        VitalWireDelay(CE_ipd, CE, tipd_CE);
        VitalWireDelay(NVSTR_ipd, NVSTR, tipd_NVSTR);
    end block;
    B1 : block
        begin
    --          EN <= SE_ipd and OE_ipd and (not ERASE_ipd) and (not PROG_ipd)
    and (not NVSTR_ipd) after 0.0001 fs;
                EN <= SE_ipd and OE_ipd and (not ERASE_ipd) and (not PROG_ipd)
    and (not NVSTR_ipd);  -- NISUNG
        end block;
    B2 : block
        begin
                Prog_Enable <= NVSTR_ipd and PROG_ipd; -- NISUNG
        end block;
    B3 : block
        begin
                Erase_Enable <= NVSTR_ipd and ERASE_ipd; -- NISUNG
        end block;
    B4: block
        begin
                Address <= XADR_ipd & YADR_ipd;
        end block;
    -------------------------------------------------------------
    -------------------------------------------------------------
    P1 : process(XADR_ipd, YADR_ipd, DIN_ipd, XE_ipd, YE_ipd, SE_ipd, OE_ipd,
    ERASE_ipd, MAS1_ipd, PROG_ipd, NVSTR_ipd, CE_ipd, Prog_Enable, Erase_Enable,
    EN, Address, Output_X_XE, Output_X_XADR, Output_X_YE, Output_X_YADR,
    Output_X_SE, Output_X_OE)
    -------------------------------------------------------------
    --  Timing Check Variable
    -------------------------------------------------------------
        variable  Tvol_DIN_YE         : std_ulogic := '0';
        variable  Tvol_DIN_YE_1       : std_ulogic := '0';
        variable  Tvol_YADR_YE        : std_ulogic := '0';
        variable  Tvol_YADR_YE_1            : std_ulogic := '0';
        variable  Tvol_ERASE_NVSTR    : std_ulogic := '0';
        variable  Tvol_NVSTR_ERASE    : std_ulogic := '0';
        variable  Tvol_NVSTR_ERASE_1  : std_ulogic := '0';
        variable  Tvol_NVSTR_ERASE_2  : std_ulogic := '0';
        variable  Tvol_MAS1_ERASE     : std_ulogic := '0';
        variable  Tvol_MAS1_NVSTR     : std_ulogic := '0';
        variable  Tvol_NVSTR_MAS1     : std_ulogic := '0';
        variable  Tvol_NVSTR_XE       : std_ulogic := '0';
        variable  Tvol_XE_ERASE       : std_ulogic := '0';
        variable  Tvol_XE_ERASE_1     : std_ulogic := '0';  -- NISUNG
        variable  Tvol_XADR_ERASE     : std_ulogic := '0';
        variable  Tvol_XADR_ERASE_1   : std_ulogic := '0';
        variable  Pvol_NVSTR          : std_ulogic := '0';
        variable  Pvol_YE             : std_ulogic := '0';
        variable  Tvol_PROG_NVSTR     : std_ulogic := '0';
```

```
        variable  Tvol_XE_NVSTR       : std_ulogic := '0';
        variable  Tvol_XADR_NVSTR     : std_ulogic := '0';
        variable  Tvol_NVSTR_YE       : std_ulogic := '0';
        variable  Tvol_NVSTR_YE_1     : std_ulogic := '0';
        variable  Tvol_PROG_YE        : std_ulogic := '0';
        variable  Tvol_XADR_PROG      : std_ulogic := '0';
        variable  Tvol_XE_PROG        : std_ulogic := '0';
        variable  Tvol_NVSTR_PROG     : std_ulogic := '0';
        variable  Tvol_NVSTR_PROG_1   : std_ulogic := '0';
        variable  TimingData_DIN_YE   : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_DIN_YE_1 : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_YADR_YE  : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_YADR_YE_1      : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_ERASE_NVSTR    : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_NVSTR_ERASE    : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_NVSTR_ERASE_1  : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_NVSTR_ERASE_2  : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_MAS1_ERASE     : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_MAS1_NVSTR     : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_NVSTR_MAS1 : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_NVSTR_XE : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_NVSTR_PROG : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_NVSTR_PROG_1 : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_XE_ERASE : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_XE_ERASE_1     : VitalTimingDataType :=
VitalTimingDataInit; -- NISUNG
        variable  TimingData_XADR_ERASE     : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_XADR_ERASE_1   : VitalTimingDataType :=
VitalTimingDataInit; -- NISUNG
        variable  TimingData_PROG_NVSTR     : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_XE_NVSTR : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_XADR_NVSTR     : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_NVSTR_YE : VitalTimingDataType :=
VitalTimingDataInit;
        variable  TimingData_NVSTR_YE_1     : VitalTimingDataType :=
VitalTimingDataInit;
```

```
      variable  TimingData_PROG_YE  : VitalTimingDataType :=
VitalTimingDataInit;
      variable  TimingData_XADR_PROG      : VitalTimingDataType :=
VitalTimingDataInit;
      variable  TimingData_XE_PROG  : VitalTimingDataType :=
VitalTimingDataInit;
      variable  PeriodData_NVSTR    : VitalPeriodDataType :=
VitalPeriodDataInit;
      variable  PeriodData_YE       : VitalPeriodDataType :=
VitalPeriodDataInit;
-------------------------------------------------------------
--  Test Mode Timing Check Variable
-------------------------------------------------------------
      variable  Tvol_NVSTR_MAS1_TestMode  : std_ulogic := '0';
      variable  TimingData_NVSTR_MAS1_TestMode  : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_NVSTR_XE_TestMode    : std_ulogic := '0';
      variable  TimingData_NVSTR_XE_TestMode    : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_NVSTR_YE_TestMode    : std_ulogic := '0';
      variable  TimingData_NVSTR_YE_TestMode    : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_NVSTR_ERASE_TestMode : std_ulogic := '0';
      variable  TimingData_NVSTR_ERASE_TestMode : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_SE_MAS1_TestMode     : std_ulogic := '0';
      variable  TimingData_SE_MAS1_TestMode     : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_SE_XE_TestMode : std_ulogic := '0';
      variable  TimingData_SE_XE_TestMode : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_SE_YE_TestMode : std_ulogic := '0';
      variable  TimingData_SE_YE_TestMode : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_SE_ERASE_TestMode    : std_ulogic := '0';
      variable  TimingData_SE_ERASE_TestMode    : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_MAS1_TMR_TestMode    : std_ulogic := '0';
      variable  TimingData_MAS1_TMR_TestMode    : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_XE_TMR_TestMode      : std_ulogic := '0';
      variable  TimingData_XE_TMR_TestMode      : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_YE_TMR_TestMode      : std_ulogic := '0';
      variable  TimingData_YE_TMR_TestMode      : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_ERASE_TMR_TestMode   : std_ulogic := '0';
      variable  TimingData_ERASE_TMR_TestMode   : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_TMR_NVSTR_TestMode   : std_ulogic := '0';
      variable  TimingData_TMR_NVSTR_TestMode   : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Tvol_TMR_SE_TestMode      : std_ulogic := '0';
      variable  TimingData_TMR_SE_TestMode      : VitalTimingDataType :=
VitalTimingDataInit;
      variable  Pvol_TMR_TestMode               : std_ulogic := '0';
```

```
      variable  PeriodData_TMR_TestMode    : VitalPeriodDataType :=
VitalPeriodDataInit;
-------------------------------------------------------------
--  Path Delay Variable
-------------------------------------------------------------
      variable  DOUT_GlitchData      : VitalGlitchDataArrayType(numOut-1
downto 0) := (others => (0 ns, 0 ns, 'X', 'X'));
      variable  DOUT_zd              : std_logic_vector(numOut-1 downto 0) :=
(others => 'X');
-------------------------------------------------------------
--  Functionality Variable
-------------------------------------------------------------
      variable MemData               : MEMORY := (others => (others => 'X'));
      variable LatchData                   : std_logic_vector(numOut-1 downto
0) := (others => 'X');
      variable EraseCounter          : COUNTER := (others => (others => '0'));
-- NISUNG
-------------------------------------------------------------
--  Other Variable
-------------------------------------------------------------
      variable EraseFlag                   : integer := 0;
--    variable ProgFlag              : integer := 0;
      variable HvTime                      : TimeArray(numRow-1 downto 0) :=
(others => 0 ns);
      variable StartEraseTime        : time := 0 ns;    -- NISUNG
      variable TotalEraseTime        : time := 0 ns;    -- NISUNG
      variable StartProgTime         : time := 0 ns;    -- NISUNG
      variable TotalProgTime         : time := 0 ns;    -- NISUNG
      variable OutLine               : line;
      variable ZeroDelay                   : VitalDelayArrayType01Z(numOut-1
downto 0) := (others =>(0 ns, 0 ns, 0 ns, 0 ns, 0 ns, 0 ns));
      variable LogicX    : std_logic_vector(numOut-1 downto 0) := (others
=>'X'); -- NISUNG
      variable TimingViolation       : std_ulogic := '0'; -- NISUNG
      variable CheckViolation        : std_ulogic := '0'; -- NISUNG
      variable Tvol_prog_mem         : std_ulogic := '0'; -- NISUNG

-------------------------------------------------------------
--    Program Memory
-------------------------------------------------------------
--    procedure Prog_Mem (                    -- NISUNG
--                      variable Violation : out std_ulogic
--                ) is
--    variable ProgData : std_logic_vector(numOut-1 downto 0) := (others =>
'X');
--    begin
--          Violation := '0';        -- NISUNG
--          if (To_Integer(Address) /= -1) then
--                ProgData := MemData(To_Integer(Address)) and LatchData ; --
MUST BE PRE-ERASED
--                ProgFlag := 1;
--                if (ProgData /= LatchData) then
--                      write(OutLine, string'("UNERASED CELL IS REPROGRAMMED
AT ")); -- NISUNG
--                      write(OutLine, Address);
--                      Write_Msg(OutLine, Error);
```

```
--                      deallocate(OutLine);
--                      Violation := '1';       -- NISUNG
--                  end if;
--                  MemData(To_Integer(Address)) := ProgData;
--          else
--                      Write_Msg("ADDRESS UNKNOWN WHEN PROGRAMMING.",
Error);
--                      Violation := '1';       -- NISUNG
--          end if;
--    end Prog_Mem;
      procedure Prog_Mem (                  -- NISUNG
                      variable Violation : out std_ulogic
                  ) is
      begin
            Violation := '0';
            if (To_Integer(Address) /= -1) then
                  if (EraseCounter(To_Integer(Address)) = "10") then
                      write(OutLine, string'("THIS CELL IS REPROGRAMMED
TWICE AT "));
                      write(OutLine, Address);
                      Write_Msg(OutLine, Error);
                      deallocate(OutLine);
                      Violation := '1';
                  end if;
                  if (EraseCounter(To_Integer(Address)) = "00") then
                      EraseCounter(To_Integer(Address)) := "01";
                  end if;
                  if (EraseCounter(To_Integer(Address)) = "01") then
                      EraseCounter(To_Integer(Address)) := "10";
                  end if;
                  MemData(To_Integer(Address)) := LatchData;
            else
                      Write_Msg("ADDRESS UNKNOWN WHEN PROGRAMMING.",
Error);
                      Violation := '1';
            end if;
      end Prog_Mem;


------------------------------------------------------------
--    Erase Memory
------------------------------------------------------------
      procedure Erase_Mem is
      variable BeginAddr      : std_logic_vector(numAddrX+numAddrY-1 downto
0) := (others => 'X');
      variable EndAddr  : std_logic_vector(numAddrX+numAddrY-1 downto 0) :=
(others => 'X');
      variable TempAddr : std_logic_vector(numAddrX+numAddrY+2 downto 0) :=
(others => 'X');
      variable TempAddr1      : std_logic_vector(numAddrX+2 downto 0) :=
(others => 'X');
      variable TempAddr2      : std_logic_vector(numAddrX+2 downto 0) :=
(others => 'X');
      variable AddrOne  : std_logic_vector(numAddrY-1 downto 0) := (others =>
'1');
```

```
      variable AddrZero : std_logic_vector(numAddrY-1 downto 0) := (others =>
'0');
      variable LogicOne       : std_logic_vector(numOut-1 downto 0) :=
(others =>'1');
      begin
            TempAddr := std_logic_vector(SHR(unsigned(XADR_ipd), "11")) &
"000" & AddrZero;
            BeginAddr := TempAddr(numAddrX+numAddrY-1 downto 0);
            TempAddr := std_logic_vector(SHR(unsigned(XADR_ipd), "11")) &
"111" & AddrOne;
            EndAddr := TempAddr(numAddrX+numAddrY-1 downto 0);
            for i in To_Integer(BeginAddr) to To_Integer(EndAddr) loop
                  MemData(i) := LogicOne;
                  EraseCounter(i) := "00";              -- NISUNG
            end loop;
            TempAddr1 := std_logic_vector(SHR(unsigned(XADR_ipd), "11")) &
"000";
            TempAddr2 := std_logic_vector(SHR(unsigned(XADR_ipd), "11")) &
"111";
            for i in To_Integer(TempAddr1) to To_Integer(TempAddr2) loop
                  HvTime(i) := 0 ns;
            end loop;
      end Erase_Mem;
-------------------------------------------------------------
--    Reset HvTime
-------------------------------------------------------------
      procedure Reset_Hvtime is
      begin
            for i in numRow-1 downto 0 loop
                  HvTime(i) := 0 ns;
            end loop;
      end Reset_Hvtime;
-------------------------------------------------------------
--    Erase Whole Memory
-------------------------------------------------------------
      procedure Erase_Whole_Mem is
      variable LogicOne : std_logic_vector(numOut-1 downto 0) := (others
=>'1');
      begin
            for i in wordDepth-1 downto 0 loop
                  MemData(i) := LogicOne;
                  EraseCounter(i) := "00";              -- NISUNG
            end loop;
            Reset_Hvtime;
      end Erase_Whole_Mem;
begin
--for i in wordDepth-1 downto 0 loop
--          MemData(i) := conv_std_logic_vector(i, numOut);
--end loop;
-------------------------------------------------------------
-- Timing Check Section
-------------------------------------------------------------
if (To_X01Z(CE_ipd) = '0') then                  -- NISUNG
      TimingData_DIN_YE := VitalTimingDataInit;
      TimingData_DIN_YE_1     := VitalTimingDataInit;
      TimingData_YADR_YE      := VitalTimingDataInit;
```

```
        TimingData_YADR_YE_1    := VitalTimingDataInit;
        TimingData_ERASE_NVSTR  := VitalTimingDataInit;
        TimingData_NVSTR_ERASE  := VitalTimingDataInit;
        TimingData_NVSTR_ERASE_1      := VitalTimingDataInit;
        TimingData_NVSTR_ERASE_2      := VitalTimingDataInit;
        TimingData_MAS1_ERASE   := VitalTimingDataInit;
        TimingData_MAS1_NVSTR   := VitalTimingDataInit;
        TimingData_NVSTR_MAS1   := VitalTimingDataInit;
        TimingData_NVSTR_XE     := VitalTimingDataInit;
        TimingData_NVSTR_PROG   := VitalTimingDataInit;
        TimingData_NVSTR_PROG_1 := VitalTimingDataInit;
        TimingData_XE_ERASE     := VitalTimingDataInit;
        TimingData_XE_ERASE_1   := VitalTimingDataInit;
        TimingData_XADR_ERASE   := VitalTimingDataInit;
        TimingData_XADR_ERASE_1 := VitalTimingDataInit;
        TimingData_PROG_NVSTR   := VitalTimingDataInit;
        TimingData_XE_NVSTR     := VitalTimingDataInit;
        TimingData_XADR_NVSTR   := VitalTimingDataInit;
        TimingData_NVSTR_YE     := VitalTimingDataInit;
        TimingData_NVSTR_YE_1   := VitalTimingDataInit;
        TimingData_PROG_YE      := VitalTimingDataInit;
        TimingData_XADR_PROG    := VitalTimingDataInit;
        TimingData_XE_PROG      := VitalTimingDataInit;
    --  Output_X_XE <= '1';
    --  Output_X_XADR     <= '1';
    --  Output_X_YE <= '1';
    --  Output_X_YADR     <= '1';
    --  Output_X_SE <= '1';
        Output_X_OE <= '1';               -- NISUNG/item 19/20
    end if;
    if (To_X01Z(CE_ipd) = '1') then
    if (TimingCheckOn) then
    ------------------------------------------------------------
    -- Check Setup timing in Erase & Mass mode
    ------------------------------------------------------------
        VitalSetupHoldCheck (
              Violation         => Tvol_ERASE_NVSTR,
              TimingData        => TimingData_ERASE_NVSTR,
              TestSignal        => ERASE_ipd,
              TestSignalName    => "ERASE",
              TestDelay         => 0 ns,
              RefSignal         => NVSTR_ipd,
              RefSignalName     => "NVSTR",
              RefDelay          => 0 ns,
              SetupHigh         => tsetup_ERASE_NVSTR_posedge_posedge,
              CheckEnabled      => true,
              RefTransition     => 'R',
              HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
              Xon                     => Xon,
              MsgOn             => MsgOn,
              MsgSeverity       => WARNING);
        VitalSetupHoldCheck (
              Violation         => Tvol_XE_NVSTR,
              TimingData        => TimingData_XE_NVSTR,
              TestSignal        => XE_ipd,
              TestSignalName    => "XE",
```

```
                TestDelay          => 0 ns,
                RefSignal          => NVSTR_ipd,
                RefSignalName      => "NVSTR",
                RefDelay           => 0 ns,
                SetupHigh          => tsetup_XE_NVSTR_posedge_posedge,
                CheckEnabled       => true,
                RefTransition      => 'R',
                HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
                Xon                      => Xon,
                MsgOn              => MsgOn,
                MsgSeverity        => WARNING);
          for i in numAddrX-1 downto 0 loop
          VitalSetupHoldCheck (
                Violation          => Tvol_XADR_NVSTR,
                TimingData         => TimingData_XADR_NVSTR,
                TestSignal         => XADR_ipd,
                TestSignalName     => "XADR",
                TestDelay          => 0 ns,
                RefSignal          => NVSTR_ipd,
                RefSignalName      => "NVSTR",
                RefDelay           => 0 ns,
                SetupHigh          => tsetup_XADR_NVSTR_noedge_posedge(i),  --
NISUNG
                SetupLow           => tsetup_XADR_NVSTR_noedge_posedge(i),  --
NISUNG

                CheckEnabled       => true,
                RefTransition      => 'R',
                HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
                Xon                      => Xon,
                MsgOn              => MsgOn,
                MsgSeverity        => WARNING);
          end loop;
    ------------------------------------------------------------
    -- Check Hold timing in Erase mode
    ------------------------------------------------------------
          VitalSetupHoldCheck (
                Violation          => Tvol_NVSTR_ERASE,
                TimingData         => TimingData_NVSTR_ERASE,
                TestSignal         => NVSTR_ipd,
                TestSignalName     => "NVSTR",
                TestDelay          => 0 ns,
                RefSignal          => ERASE_ipd,
                RefSignalName      => "ERASE",
                RefDelay           => 0 ns,
                HoldHigh           => thold_NVSTR_ERASE_negedge_negedge,
                CheckEnabled       => (To_X01Z(MAS1_ipd) = '0'),
                RefTransition      => 'F',
                HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
                Xon                      => Xon,
                MsgOn              => MsgOn,
                MsgSeverity        => WARNING);
          VitalSetupHoldCheck (
                Violation          => Tvol_XE_ERASE,
                TimingData         => TimingData_XE_ERASE,
                TestSignal         => XE_ipd,
```

```
                TestSignalName     => "XE",
                TestDelay          => 0 ns,
                RefSignal          => ERASE_ipd,
                RefSignalName      => "ERASE",
                RefDelay           => 0 ns,
                HoldHigh           => thold_XE_ERASE_negedge_negedge,
                CheckEnabled       => (To_X01Z(MAS1_ipd) = '0'),
                RefTransition      => 'F',
                HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
                Xon                      => Xon,
                MsgOn              => MsgOn,
                MsgSeverity        => WARNING);
        for i in numAddrX-1 downto 0 loop
        VitalSetupHoldCheck (
                Violation          => Tvol_XADR_ERASE,
                TimingData         => TimingData_XADR_ERASE,
                TestSignal         => XADR_ipd,
                TestSignalName     => "XADR",
                TestDelay          => 0 ns,
                RefSignal          => ERASE_ipd,
                RefSignalName      => "ERASE",
                RefDelay           => 0 ns,
                HoldHigh           => thold_XADR_ERASE_negedge_negedge(i),
                CheckEnabled       => (To_X01Z(MAS1_ipd) = '0'),
                RefTransition      => 'F',
                HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
                Xon                      => Xon,
                MsgOn              => MsgOn,
                MsgSeverity        => WARNING);
        end loop;
    ------------------------------------------------------------
    -- Check mininal pulse width in Erase mode
    ------------------------------------------------------------
        VitalPeriodPulseCheck (
                Violation          => Pvol_NVSTR,
                PeriodData         => PeriodData_NVSTR,
                TestSignal         => NVSTR_ipd,
                TestSignalName     => "NVSTR",
                TestDelay          => 0 ns,
                PulseWidthHigh     => tpw_NVSTR_posedge,
                CheckEnabled       => ((To_X01Z(Erase_Enable) = '1') and
                                       (To_X01Z(MAS1_ipd) = '0')),
                HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
                Xon                      => Xon,
                MsgOn              => MsgOn,
                MsgSeverity        => WARNING);

    ------------------------------------------------------------
    -- Check recovery time Erase & Mass mode
    ------------------------------------------------------------
        VitalRecoveryRemovalCheck (
                Violation               => Tvol_NVSTR_MAS1,
                TimingData              => TimingData_NVSTR_MAS1,
                TestSignal              => NVSTR_ipd,
                TestSignalName          => "NVSTR",
                TestDelay               => 0 ns,
```

```
                RefSignal               => MAS1_ipd,
                RefSignalName           => "MAS1",
                RefDelay                => 0 ns,
                Recovery                => trecovery_NVSTR_MAS1_negedge_posedge,
                ActiveLow               => false,           -- TestSignal from 1 -
> 0
                CheckEnabled            => true,
                RefTransition           => 'R',             -- RefSignal from 0 ->
1
                HeaderMsg               => InstancePath &
"/SFB0008_08B9_CORE_MAIN",
                Xon                     => Xon,
                MsgOn                   => MsgOn,
                MsgSeverity             => WARNING);
        VitalRecoveryRemovalCheck (
                Violation               => Tvol_NVSTR_XE,
                TimingData              => TimingData_NVSTR_XE,
                TestSignal              => NVSTR_ipd,
                TestSignalName          => "NVSTR",
                TestDelay               => 0 ns,
                RefSignal               => XE_ipd,
                RefSignalName           => "XE",
                RefDelay                => 0 ns,
                Recovery                => trecovery_NVSTR_XE_negedge_posedge,
                ActiveLow               => false,           -- TestSignal from 1 -
> 0
                CheckEnabled            => true,
                RefTransition           => 'R',             -- RefSignal from 0 ->
1
                HeaderMsg               => InstancePath &
"/SFB0008_08B9_CORE_MAIN",
                Xon                     => Xon,
                MsgOn                   => MsgOn,
                MsgSeverity             => WARNING);
        VitalRecoveryRemovalCheck (
                Violation               => Tvol_NVSTR_ERASE_1,
                TimingData              => TimingData_NVSTR_ERASE_1,
                TestSignal              => NVSTR_ipd,
                TestSignalName          => "NVSTR",
                TestDelay               => 0 ns,
                RefSignal               => ERASE_ipd,
                RefSignalName           => "ERASE",
                RefDelay                => 0 ns,
                Recovery                => trecovery_NVSTR_ERASE_negedge_posedge,
                ActiveLow               => false,           -- TestSignal from 1 -
> 0
                CheckEnabled            => true,
                RefTransition           => 'R',             -- RefSignal from 0 ->
1
                HeaderMsg               => InstancePath &
"/SFB0008_08B9_CORE_MAIN",
                Xon                     => Xon,
                MsgOn                   => MsgOn,
                MsgSeverity             => WARNING);
-------------------------------------------------------------
-- Check Setup/Hold timing in Mass Erase mode
```

```
-------------------------------------------------------------
      VitalSetupHoldCheck (                  -- NISUNG
            Violation          => Tvol_MAS1_NVSTR,
            TimingData         => TimingData_MAS1_NVSTR,
            TestSignal         => MAS1_ipd,
            TestSignalName     => "MAS1",
            TestDelay          => 0 ns,
            RefSignal          => NVSTR_ipd,
            RefSignalName      => "NVSTR",
            RefDelay           => 0 ns,
            SetupHigh          => tsetup_MAS1_NVSTR_posedge_posedge,
            CheckEnabled       => (To_X01Z(MAS1_ipd) = '1'),
            RefTransition      => 'R',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                      => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      VitalSetupHoldCheck (           -- NISUNG
            Violation          => Tvol_NVSTR_ERASE_2,
            TimingData         => TimingData_NVSTR_ERASE_2,
            TestSignal         => NVSTR_ipd,
            TestSignalName     => "NVSTR",
            TestDelay          => 0 ns,
            RefSignal          => ERASE_ipd,
            RefSignalName      => "ERASE",
            RefDelay           => 0 ns,
            HoldHigh           => Tnvh1,
            CheckEnabled       => (To_X01Z(MAS1_ipd) = '1'),
            RefTransition      => 'F',
            HeaderMsg          => InstancePath &
"/SFB0008_08B9_CORE_MAIN/Tnvh1",
            Xon                      => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      for i in numAddrX-1 downto 0 loop
      VitalSetupHoldCheck (               -- NISUNG
            Violation          => Tvol_XADR_ERASE_1,
            TimingData         => TimingData_XADR_ERASE_1,
            TestSignal         => XADR_ipd,
            TestSignalName     => "XADR",
            TestDelay          => 0 ns,
            RefSignal          => ERASE_ipd,
            RefSignalName      => "ERASE",
            RefDelay           => 0 ns,
            HoldHigh           => Tnvh1,
            CheckEnabled       => (To_X01Z(MAS1_ipd) = '1'),
            RefTransition      => 'F',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_INFO",
            Xon                      => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      end loop;
      VitalSetupHoldCheck (               -- NISUNG
            Violation          => Tvol_XE_ERASE_1,
            TimingData         => TimingData_XE_ERASE_1,
            TestSignal         => XE_ipd,
```

```
                TestSignalName     => "XE",
                TestDelay          => 0 ns,
                RefSignal          => ERASE_ipd,
                RefSignalName      => "ERASE",
                RefDelay           => 0 ns,
                HoldHigh           => Tnvh1,
                CheckEnabled       => (To_X01Z(MAS1_ipd) = '1'),
                RefTransition      => 'F',
                HeaderMsg          => InstancePath &
"/SFB0008_08B9_CORE_MAIN/Tnvh1",
                Xon                      => Xon,
                MsgOn              => MsgOn,
                MsgSeverity        => WARNING);
        VitalSetupHoldCheck (                    -- NISUNG
                Violation          => Tvol_MAS1_ERASE,
                TimingData         => TimingData_MAS1_ERASE,
                TestSignal         => MAS1_ipd,
                TestSignalName     => "MAS1",
                TestDelay          => 0 ns,
                RefSignal          => ERASE_ipd,
                RefSignalName      => "ERASE",
                RefDelay           => 0 ns,
                HoldHigh           => thold_MAS1_ERASE_negedge_negedge,
                CheckEnabled       => (To_X01Z(MAS1_ipd) = '0'),
                RefTransition      => 'F',
                HeaderMsg          => InstancePath &
"/SFB0008_08B9_CORE_MAIN/Tnvh1",
                Xon                      => Xon,
                MsgOn              => MsgOn,
                MsgSeverity        => WARNING);
    ------------------------------------------------------------
    -- Check mininal pulse width in Mass Erase mode
    ------------------------------------------------------------
        VitalPeriodPulseCheck (
                Violation          => Pvol_NVSTR,
                PeriodData         => PeriodData_NVSTR,
                TestSignal         => NVSTR_ipd,
                TestSignalName     => "NVSTR",
                TestDelay          => 0 ns,
                PulseWidthHigh     => Tme,
                CheckEnabled       => ((To_X01Z(Erase_Enable) = '1') and
                                          (To_X01Z(MAS1_ipd) = '1')),
                HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
                Xon                      => Xon,
                MsgOn              => MsgOn,
                MsgSeverity        => WARNING);

    ------------------------------------------------------------
    -- Check Setup/Hold timing in PROG mode
    ------------------------------------------------------------
        VitalSetupHoldCheck (
                Violation          => Tvol_PROG_NVSTR,
                TimingData         => TimingData_PROG_NVSTR,
                TestSignal         => PROG_ipd,
                TestSignalName     => "PROG",
                TestDelay          => 0 ns,
```

```
            RefSignal          => NVSTR_ipd,
            RefSignalName      => "NVSTR",
            RefDelay           => 0 ns,
            SetupHigh          => tsetup_PROG_NVSTR_posedge_posedge,
            CheckEnabled       => true,
            RefTransition      => 'R',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                      => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      VitalSetupHoldCheck (
            Violation          => Tvol_NVSTR_YE,
            TimingData         => TimingData_NVSTR_YE,
            TestSignal         => NVSTR_ipd,
            TestSignalName     => "NVSTR",
            TestDelay          => 0 ns,
            RefSignal          => YE_ipd,
            RefSignalName      => "YE",
            RefDelay           => 0 ns,
            SetupHigh          => tsetup_NVSTR_YE_posedge_posedge,
            CheckEnabled       => (To_X01Z(Prog_Enable) = '1'),
            RefTransition      => 'R',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                      => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      for i in numOut-1 downto 0 loop
      VitalSetupHoldCheck (
            Violation          => Tvol_DIN_YE,
            TimingData         => TimingData_DIN_YE,
            TestSignal         => DIN_ipd,
            TestSignalName     => "Bus DIN",
            TestDelay          => 0 ns,
            RefSignal          => YE_ipd,
            RefSignalName      => "YE",
            RefDelay           => 0 ns,
            SetupHigh          => tsetup_DIN_YE_noedge_posedge(i),
            SetupLow           => tsetup_DIN_YE_noedge_posedge(i),
            CheckEnabled       => (To_X01Z(Prog_Enable) = '1'),
            RefTransition      => 'R',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                      => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      end loop;
      for i in numOut-1 downto 0 loop
      VitalSetupHoldCheck (
            Violation          => Tvol_DIN_YE_1,
            TimingData         => TimingData_DIN_YE_1,
            TestSignal         => DIN_ipd,
            TestSignalName     => "Bus DIN",
            TestDelay          => 0 ns,
            RefSignal          => YE_ipd,
            RefSignalName      => "YE",
            RefDelay           => 0 ns,
            HoldHigh           => thold_DIN_YE_noedge_negedge(i),
```

```
            HoldLow                 => thold_DIN_YE_noedge_negedge(i),
            CheckEnabled      => (To_X01Z(Prog_Enable) = '1'),
            RefTransition     => 'F',
            HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn             => MsgOn,
            MsgSeverity       => WARNING);
      end loop;
      for i in numAddrY-1 downto 0 loop
      VitalSetupHoldCheck (
            Violation         => Tvol_YADR_YE,
            TimingData        => TimingData_YADR_YE,
            TestSignal        => YADR_ipd,
            TestSignalName    => "Bus YADR",
            TestDelay         => 0 ns,
            RefSignal         => YE_ipd,
            RefSignalName     => "YE",
            RefDelay          => 0 ns,
            SetupHigh         => tsetup_YADR_YE_noedge_posedge(i),
            SetupLow          => tsetup_YADR_YE_noedge_posedge(i),
            CheckEnabled      => (To_X01Z(Prog_Enable) = '1'),
            RefTransition     => 'R',
            HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn             => MsgOn,
            MsgSeverity       => WARNING);
      end loop;
      for i in numAddrY-1 downto 0 loop
      VitalSetupHoldCheck (
            Violation         => Tvol_YADR_YE_1,
            TimingData        => TimingData_YADR_YE_1,
            TestSignal        => YADR_ipd,
            TestSignalName    => "Bus YADR",
            TestDelay         => 0 ns,
            RefSignal         => YE_ipd,
            RefSignalName     => "YE",
            RefDelay          => 0 ns,
            HoldHigh          => thold_YADR_YE_noedge_negedge(i),
            HoldLow                 => thold_YADR_YE_noedge_negedge(i),
            CheckEnabled      => (To_X01Z(Prog_Enable) = '1'),
            RefTransition     => 'F',
            HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn             => MsgOn,
            MsgSeverity       => WARNING);
      end loop;
      VitalSetupHoldCheck (
            Violation         => Tvol_PROG_YE,
            TimingData        => TimingData_PROG_YE,
            TestSignal        => PROG_ipd,
            TestSignalName    => "PROG",
            TestDelay         => 0 ns,
            RefSignal         => YE_ipd,
            RefSignalName     => "YE",
            RefDelay          => 0 ns,
            HoldHigh          => thold_PROG_YE_negedge_negedge,
```

```
            CheckEnabled      => (To_X01Z(Prog_Enable) = '1'),
            RefTransition     => 'F',
            HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                  => Xon,
            MsgOn             => MsgOn,
            MsgSeverity       => WARNING);
      for i in numAddrX-1 downto 0 loop
      VitalSetupHoldCheck (
            Violation         => Tvol_XADR_PROG,
            TimingData        => TimingData_XADR_PROG,
            TestSignal        => XADR_ipd,
            TestSignalName    => "XADR",
            TestDelay         => 0 ns,
            RefSignal         => PROG_ipd,
            RefSignalName     => "PROG",
            RefDelay          => 0 ns,
            HoldHigh          => thold_XADR_PROG_negedge_negedge(i),
            CheckEnabled      => true,
            RefTransition     => 'F',
            HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                  => Xon,
            MsgOn             => MsgOn,
            MsgSeverity       => WARNING);
      end loop;
      VitalSetupHoldCheck (
            Violation         => Tvol_XE_PROG,
            TimingData        => TimingData_XE_PROG,
            TestSignal        => XE_ipd,
            TestSignalName    => "XE",
            TestDelay         => 0 ns,
            RefSignal         => PROG_ipd,
            RefSignalName     => "PROG",
            RefDelay          => 0 ns,
            HoldHigh          => thold_XE_PROG_negedge_negedge,
            CheckEnabled      => true,
            RefTransition     => 'F',
            HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                  => Xon,
            MsgOn             => MsgOn,
            MsgSeverity       => WARNING);
      VitalSetupHoldCheck (
            Violation         => Tvol_NVSTR_PROG,
            TimingData        => TimingData_NVSTR_PROG,
            TestSignal        => NVSTR_ipd,
            TestSignalName    => "NVSTR",
            TestDelay         => 0 ns,
            RefSignal         => PROG_ipd,
            RefSignalName     => "PROG",
            RefDelay          => 0 ns,
            HoldHigh          => thold_NVSTR_PROG_negedge_negedge,
            CheckEnabled      => true,
            RefTransition     => 'F',
            HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                  => Xon,
            MsgOn             => MsgOn,
            MsgSeverity       => WARNING);
```

```
-------------------------------------------------------------
-- Check recovery time Prog mode
-------------------------------------------------------------
      VitalRecoveryRemovalCheck (
            Violation               => Tvol_NVSTR_PROG_1,
            TimingData              => TimingData_NVSTR_PROG_1,
            TestSignal              => NVSTR_ipd,
            TestSignalName          => "NVSTR",
            TestDelay               => 0 ns,
            RefSignal               => PROG_ipd,
            RefSignalName           => "PROG",
            RefDelay                => 0 ns,
            Recovery                => trecovery_NVSTR_PROG_negedge_posedge,
            ActiveLow               => false,
            CheckEnabled            => TRUE,
            RefTransition           => 'R',
            HeaderMsg               => InstancePath &
"/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn                   => MsgOn,
            MsgSeverity             => WARNING);
      VitalRecoveryRemovalCheck (
            Violation               => Tvol_NVSTR_YE_1,
            TimingData              => TimingData_NVSTR_YE_1,
            TestSignal              => NVSTR_ipd,
            TestSignalName          => "NVSTR",
            TestDelay               => 0 ns,
            RefSignal               => YE_ipd,
            RefSignalName           => "YE",
            RefDelay                => 0 ns,
            Recovery                => trecovery_NVSTR_YE_negedge_posedge,
            ActiveLow               => false,
            CheckEnabled            => TRUE,
            RefTransition           => 'R',
            HeaderMsg               => InstancePath &
"/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn                   => MsgOn,
            MsgSeverity             => WARNING);
-------------------------------------------------------------
-- Check mininal pulse width in Prog mode
-------------------------------------------------------------
      VitalPeriodPulseCheck (
            Violation         => Pvol_YE,
            PeriodData        => PeriodData_YE,
            TestSignal        => YE_ipd,
            TestSignalName    => "YE",
            TestDelay         => 0 ns,
            PulseWidthHigh    => tpw_YE_posedge,
            CheckEnabled      => To_X01Z(Prog_Enable) = '1',
            HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                  => Xon,
            MsgOn             => MsgOn,
            MsgSeverity       => WARNING);
-------------------------------------------------------------
-- Test Mode timing check
```

```
-------------------------------------------------------------
      VitalSetupHoldCheck (
            Violation          => Tvol_NVSTR_MAS1_TestMode,
            TimingData         => TimingData_NVSTR_MAS1_TestMode,
            TestSignal         => NVSTR_ipd,
            TestSignalName     => "NVSTR",
            TestDelay          => 0 ns,
            RefSignal          => MAS1_ipd,
            RefSignalName      => "MAS1",
            RefDelay           => 0 ns,
            SetupHigh          => Tseds,
            HoldLow                 => Tdseh,
            CheckEnabled       => (To_X01Z(TMR) = '0'),
            RefTransition      => 'R',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      VitalSetupHoldCheck (
            Violation          => Tvol_NVSTR_XE_TestMode,
            TimingData         => TimingData_NVSTR_XE_TestMode,
            TestSignal         => NVSTR_ipd,
            TestSignalName     => "NVSTR",
            TestDelay          => 0 ns,
            RefSignal          => XE_ipd,
            RefSignalName      => "XE",
            RefDelay           => 0 ns,
            SetupHigh          => Tseds,
            HoldLow                 => Tdseh,
            CheckEnabled       => (To_X01Z(TMR) = '0'),
            RefTransition      => 'R',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      VitalSetupHoldCheck (
            Violation          => Tvol_NVSTR_YE_TestMode,
            TimingData         => TimingData_NVSTR_YE_TestMode,
            TestSignal         => NVSTR_ipd,
            TestSignalName     => "NVSTR",
            TestDelay          => 0 ns,
            RefSignal          => YE_ipd,
            RefSignalName      => "YE",
            RefDelay           => 0 ns,
            SetupHigh          => Tseds,
            HoldLow                 => Tdseh,
            CheckEnabled       => (To_X01Z(TMR) = '0'),
            RefTransition      => 'R',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      VitalSetupHoldCheck (
            Violation          => Tvol_NVSTR_ERASE_TestMode,
            TimingData         => TimingData_NVSTR_ERASE_TestMode,
            TestSignal         => NVSTR_ipd,
```

```
            TestSignalName    => "NVSTR",
            TestDelay         => 0 ns,
            RefSignal         => ERASE_ipd,
            RefSignalName     => "ERASE",
            RefDelay          => 0 ns,
            SetupHigh         => Tseds,
            HoldLow                 => Tdseh,
            CheckEnabled      => (To_X01Z(TMR) = '0'),
            RefTransition     => 'R',
            HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn             => MsgOn,
            MsgSeverity       => WARNING);
        VitalSetupHoldCheck (
            Violation         => Tvol_SE_MAS1_TestMode,
            TimingData        => TimingData_SE_MAS1_TestMode,
            TestSignal        => SE_ipd,
            TestSignalName    => "SE",
            TestDelay         => 0 ns,
            RefSignal         => MAS1_ipd,
            RefSignalName     => "MAS1",
            RefDelay          => 0 ns,
            SetupHigh         => Tseds,
            HoldLow                 => Tdseh,
            CheckEnabled      => (To_X01Z(TMR) = '0'),
            RefTransition     => 'R',
            HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn             => MsgOn,
            MsgSeverity       => WARNING);
        VitalSetupHoldCheck (
            Violation         => Tvol_SE_XE_TestMode,
            TimingData        => TimingData_SE_XE_TestMode,
            TestSignal        => SE_ipd,
            TestSignalName    => "SE",
            TestDelay         => 0 ns,
            RefSignal         => XE_ipd,
            RefSignalName     => "XE",
            RefDelay          => 0 ns,
            SetupHigh         => Tseds,
            HoldLow                 => Tdseh,
            CheckEnabled      => (To_X01Z(TMR) = '0'),
            RefTransition     => 'R',
            HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn             => MsgOn,
            MsgSeverity       => WARNING);
        VitalSetupHoldCheck (
            Violation         => Tvol_SE_YE_TestMode,
            TimingData        => TimingData_SE_YE_TestMode,
            TestSignal        => SE_ipd,
            TestSignalName    => "SE",
            TestDelay         => 0 ns,
            RefSignal         => YE_ipd,
            RefSignalName     => "YE",
            RefDelay          => 0 ns,
```

```
            SetupHigh          => Tseds,
            HoldLow                 => Tdseh,
            CheckEnabled       => (To_X01Z(TMR) = '0'),
            RefTransition      => 'R',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      VitalSetupHoldCheck (
            Violation          => Tvol_SE_ERASE_TestMode,
            TimingData         => TimingData_SE_ERASE_TestMode,
            TestSignal         => SE_ipd,
            TestSignalName     => "SE",
            TestDelay          => 0 ns,
            RefSignal          => ERASE_ipd,
            RefSignalName      => "ERASE",
            RefDelay           => 0 ns,
            SetupHigh          => Tseds,
            HoldLow                 => Tdseh,
            CheckEnabled       => (To_X01Z(TMR) = '0'),
            RefTransition      => 'R',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      VitalSetupHoldCheck (
            Violation          => Tvol_MAS1_TMR_TestMode,
            TimingData         => TimingData_MAS1_TMR_TestMode,
            TestSignal         => MAS1_ipd,
            TestSignalName     => "MAS1",
            TestDelay          => 0 ns,
            RefSignal          => TMR,
            RefSignalName      => "TMR",
            RefDelay           => 0 ns,
            SetupHigh          => Tlds,
            HoldLow                 => Tldh,
            CheckEnabled       => (To_X01Z(TMR) = '0'),
            RefTransition      => 'R',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
            Xon                     => Xon,
            MsgOn              => MsgOn,
            MsgSeverity        => WARNING);
      VitalSetupHoldCheck (
            Violation          => Tvol_XE_TMR_TestMode,
            TimingData         => TimingData_XE_TMR_TestMode,
            TestSignal         => XE_ipd,
            TestSignalName     => "XE",
            TestDelay          => 0 ns,
            RefSignal          => TMR,
            RefSignalName      => "TMR",
            RefDelay           => 0 ns,
            SetupHigh          => Tlds,
            HoldLow                 => Tldh,
            CheckEnabled       => (To_X01Z(TMR) = '0'),
            RefTransition      => 'R',
            HeaderMsg          => InstancePath & "/SFB0008_08B9_CORE_MAIN",
```

```
                Xon                     => Xon,
                MsgOn                   => MsgOn,
                MsgSeverity             => WARNING);
        VitalSetupHoldCheck (
                Violation               => Tvol_ERASE_TMR_TestMode,
                TimingData              => TimingData_ERASE_TMR_TestMode,
                TestSignal              => ERASE_ipd,
                TestSignalName          => "ERASE",
                TestDelay               => 0 ns,
                RefSignal               => TMR,
                RefSignalName           => "TMR",
                RefDelay                => 0 ns,
                SetupHigh               => Tlds,
                HoldLow                 => Tldh,
                CheckEnabled            => (To_X01Z(TMR) = '0'),
                RefTransition           => 'R',
                HeaderMsg               => InstancePath & "/SFB0008_08B9_CORE_MAIN",
                Xon                     => Xon,
                MsgOn                   => MsgOn,
                MsgSeverity             => WARNING);
        VitalSetupHoldCheck (
                Violation               => Tvol_XE_TMR_TestMode,
                TimingData              => TimingData_XE_TMR_TestMode,
                TestSignal              => XE_ipd,
                TestSignalName          => "XE",
                TestDelay               => 0 ns,
                RefSignal               => TMR,
                RefSignalName           => "TMR",
                RefDelay                => 0 ns,
                SetupHigh               => Tlds,
                HoldLow                 => Tldh,
                CheckEnabled            => (To_X01Z(TMR) = '0'),
                RefTransition           => 'R',
                HeaderMsg               => InstancePath & "/SFB0008_08B9_CORE_MAIN",
                Xon                     => Xon,
                MsgOn                   => MsgOn,
                MsgSeverity             => WARNING);
        VitalRecoveryRemovalCheck (
                Violation               => Tvol_TMR_NVSTR_TestMode,
                TimingData              => TimingData_TMR_NVSTR_TestMode,
                TestSignal              => TMR,
                TestSignalName          => "TMR",
                TestDelay               => 0 ns,
                RefSignal               => NVSTR_ipd,
                RefSignalName           => "NVSTR",
                RefDelay                => 0 ns,
                Recovery                => Trses,
                ActiveLow               => false,
                CheckEnabled            => (To_X01Z(TMR) = '0'),
                RefTransition           => 'R',
                HeaderMsg               => InstancePath &
    "/SFB0008_08B9_CORE_MAIN",
                Xon                     => Xon,
                MsgOn                   => MsgOn,
                MsgSeverity             => WARNING);
        VitalRecoveryRemovalCheck (
```

```
                Violation               => Tvol_TMR_SE_TestMode,
                TimingData              => TimingData_TMR_SE_TestMode,
                TestSignal              => TMR,
                TestSignalName          => "TMR",
                TestDelay               => 0 ns,
                RefSignal               => SE_ipd,
                RefSignalName           => "SE",
                RefDelay                => 0 ns,
                Recovery                => Trses,
                ActiveLow               => false,
                CheckEnabled                 => (To_X01Z(TMR) = '0'),
                RefTransition           => 'R',
                HeaderMsg               => InstancePath &
"/SFB0008_08B9_CORE_MAIN",
                Xon                     => Xon,
                MsgOn                   => MsgOn,
                MsgSeverity             => WARNING);
        VitalPeriodPulseCheck (
                Violation         => Pvol_TMR_TestMode,
                PeriodData        => PeriodData_TMR_TestMode,
                TestSignal        => TMR,
                TestSignalName    => "TMR",
                TestDelay         => 0 ns,
                PulseWidthHigh    => Tlpw,
                CheckEnabled      => (To_X01Z(TMR) = '0'),
                HeaderMsg         => InstancePath & "/SFB0008_08B9_CORE_MAIN",
                Xon                    => Xon,
                MsgOn             => MsgOn,
                MsgSeverity       => WARNING);
end if;
-------------------------------------------------------------
-- Functionality
-------------------------------------------------------------
TimingViolation := Tvol_DIN_YE or Tvol_DIN_YE_1 or Tvol_YADR_YE or
                   Tvol_YADR_YE_1 or Tvol_ERASE_NVSTR or Tvol_NVSTR_ERASE or
                   Tvol_NVSTR_ERASE_1 or Tvol_NVSTR_ERASE_2 or Tvol_MAS1_ERASE
      or
             Tvol_MAS1_NVSTR   or Tvol_NVSTR_MAS1 or Tvol_NVSTR_XE or
             Tvol_XE_ERASE or Tvol_XE_ERASE_1 or Tvol_XADR_ERASE or
                   Tvol_XADR_ERASE_1 or Pvol_NVSTR or
             Pvol_YE or Tvol_PROG_NVSTR or Tvol_XE_NVSTR or
             Tvol_XADR_NVSTR   or Tvol_NVSTR_YE or Tvol_NVSTR_YE_1 or
             Tvol_PROG_YE or Tvol_XADR_PROG or Tvol_XE_PROG or
             Tvol_NVSTR_PROG   or Tvol_NVSTR_PROG or CheckViolation or
                   Tvol_prog_mem or P3Violation;   -- NISUNG/item 32
if ((DebugMode = False) and (TimingViolation = '1')) then
      for i in wordDepth-1 downto 0 loop
            MemData(i) := LogicX;
      end loop;
end if;
CheckViolation := '0';
if (NOW /= 0 ns) then
-------------------------------------------------------------
-- Program Cycle
-------------------------------------------------------------
      if ((YADR_ipd'event) or (YE_ipd'event)) then
```

```
                if (To_X01Z(Prog_Enable) = '1') then
                      if ((To_X01Z(YE_ipd) = '1') and
                            (To_X01Z(YE_ipd'last_value) = '0')) then
                            LatchData := DIN_ipd;
5   --                      ProgFlag := 1;
                            Prog_Mem(                              -- NISUNG
                                  Violation => Tvol_prog_mem);
    --                elsif ((To_X01Z(YE_ipd) = '0') and
    --
10  --                      (To_X01Z(YE_ipd'last_value) = '1') and
    --                      (ProgFlag = 1)) then
                          ProgFlag := 0;
                      end if;
                end if;
          end if;
15  ------------------------------------------------------------
    -- Check Cumulative delay Thv
    ------------------------------------------------------------
          if (Prog_Enable'event) then
                if (To_X01Z(Prog_Enable) = '1') then
20                    StartProgTime := NOW;
                else
                      TotalProgTime :=
    HvTime(To_Integer(std_logic_vector(unsigned(XADR_ipd)))) + NOW -
    StartProgTime;
25                    if (TotalProgTime > Thv) then
                            Write(OutLine, string'("PROGRAM HIGH VOLTAGE TIME
    ("));
                            Write(OutLine, TotalProgTime);
                            Write(OutLine, string'(") TOO LONG ! > "));
30                          Write(OutLine, Thv);
                            Write_Msg(OutLine, Error);
                            deallocate(OutLine);
                            CheckViolation := '1';       -- NISUNG
                      end if;
35                    HvTime(To_Integer(std_logic_vector(unsigned(XADR_ipd)))) :=
    TotalProgTime ;
                end if;
          end if;
    ------------------------------------------------------------
40  -- Erase Cycle
    ------------------------------------------------------------
          if (Erase_Enable'event) then
                if (To_X01Z(Erase_Enable) = '1') then
                      EraseFlag := 1;
45                    if (To_X01Z(MAS1_ipd) = '1') then
                            Write_Msg(" ERASING ENTIRE MEMORY...", Note); --
    NISUNG
                            Erase_Whole_Mem;
                      else
50                          Write(OutLine, string'(" ERASING PAGE => "));
                            Write(OutLine,
    std_logic_vector(SHR(unsigned(XADR_ipd), "11")));
                            Write_Msg(OutLine, Note);  -- NISUNG
                            deallocate(OutLine);
55                          if (To_X01Z(XE_ipd) /= '1') then          -- NISUNG
```

```
                                    Write_Msg("XE IS NOT HIGH DURING ERASE.",
Error);
                                    EraseFlag := 0;
                                    CheckViolation := '1';        -- NISUNG
                              end if;
                              if (To_Integer(XADR_ipd) = -1) then
                                    Write_Msg("X ADDRESS IS UNKNOWN.", Error);
                                    EraseFlag := 0;
                                    CheckViolation := '1';        -- NISUNG
                              end if;
                              if (EraseFlag = 1) then
                                    Erase_Mem;
                              end if;
                        end if;
                        StartEraseTime := NOW;
                  elsif (To_X01Z(Erase_Enable) = '0') then
                        if (EraseFlag = 1) then
                              TotalEraseTime := NOW - StartEraseTime;
                              if (To_X01Z(MAS1_ipd) = '1') then
                                    if (TotalEraseTime < Tme ) then
                                          Write(OutLine, string'("MASS ERASE PULSE
("));
                                          Write(OutLine, TotalEraseTime);
                                          Write(OutLine, string'(") TOO SHORT ! <
"));
                                          Write(OutLine, Tme);
                                          Write_Msg(OutLine, Error);
                                          deallocate(OutLine);
                                          CheckViolation := '1';        -- NISUNG
                                    else
                                          Write_Msg(" MASS ERASE COMPLETE !",
Note);
                                    end if;
                              else
                                    if (TotalEraseTime < Terase ) then
                                          Write(OutLine, string'("ERASE PULSE ("));
                                          Write(OutLine, TotalEraseTime);
                                          Write(OutLine, string'(") TOO SHORT ! <
"));
                                          Write(OutLine, Terase);
                                          Write_Msg(OutLine, Error);
                                          deallocate(OutLine);
                                          CheckViolation := '1';        -- NISUNG
                                    end if;
                              end if;
                        end if;
                        EraseFlag := 0;
                  end if;
            end if;
      ------------------------------------------------------------
      -- Check MAS1
      ------------------------------------------------------------
            if (MAS1_ipd'event) then
                  if (To_X01Z(MAS1_ipd) = 'X') then
                        Write_Msg("MAS1 IS UNKNOWN !", Warning);
```

```
                    CheckViolation := '1';        -- NISUNG
                end if;
                if (EraseFlag = 1) then
                    Write_Msg("MAS1 CAN NOT CHANGE DURING ERASE CYCLE !",
Error);
                    CheckViolation := '1';        -- NISUNG
                end if;
        end if;
-------------------------------------------------------------
-- Read Cycle & Path Delay Section
-------------------------------------------------------------
        if (((XE_ipd'event) and (To_X01Z(XE_ipd) = '1')) or
            ((CE_ipd'event) and (To_X01Z(XE_ipd) = '1'))) then
              Output_X_XE <= '1' after 0 ns, '0' after tpd_XE_DOUT(0)(tr01) ;
        end if;
        if ((XE_ipd'event) and (To_X01Z(XE_ipd) = '0')) then
              Output_X_XE <= '1' after 0 ns;
        end if;
        if (XADR_ipd'event) then
                    Output_X_XADR <= '1' after 0 ns, '0' after
tpd_XE_DOUT(0)(tr01) ;
        end if;
        if (((YE_ipd'event) and (To_X01Z(YE_ipd) = '1')) or
            ((CE_ipd'event) and (To_X01Z(YE_ipd) = '1'))) then
              Output_X_YE <= '1' after 0 ns, '0' after tpd_YE_DOUT(0)(tr01) ;
        end if;
        if ((YE_ipd'event) and (To_X01Z(YE_ipd) = '0')) then
              Output_X_YE <= transport '1' after 0 ns;
        end if;
        if (YADR_ipd'event) then
                    Output_X_YADR <= '1' after 0 ns, '0' after
tpd_YE_DOUT(0)(tr01) ;
        end if;
        if ((SE_ipd'event) and (To_X01Z(SE_ipd) = '1')) then
              Output_X_SE <= '1' after 0 ns, '0' after tpd_SE_DOUT(0)(tr01) ;
        end if;
        if ((OE_ipd'event) and (To_X01Z(OE_ipd) = '1')) then
              Output_X_OE <= 'Z' after 0 ns, '0' after tpd_OE_DOUT(0)(tr0Z) ;
        end if;
        if ((OE_ipd'event) and (To_X01Z(OE_ipd) = '0')) then
              Output_X_OE <= 'Z' after 0 ns;
        end if;

        if (To_X01Z(EN) = '0') then                          -- NISUNG
              if (To_X01Z(OE_ipd) = '1') then
                    if (To_X01Z(SE_ipd) = '0') then
                          for i in DOUT_zd'range loop
                                DOUT_zd(i) := '0';
                          end loop;
                    else
                          for i in DOUT_zd'range loop
                                DOUT_zd(i) := 'X';
                          end loop;
                    end if;
              else
                    for i in DOUT_zd'range loop
```

```
                        DOUT_zd(i) := 'Z';
                    end loop;
                end if;
            end if;                                                    -- END
            if (To_X01Z(EN) = '1') then
                if (To_X01Z(Output_X_OE) = 'Z') then
                    for i in DOUT_zd'range loop
                        DOUT_zd(i) := 'Z';
                    end loop;
                elsif  ((To_X01Z(Output_X_XE) = '1') or
                    (To_X01Z(Output_X_XADR) = '1') or
                    (To_X01Z(Output_X_YE) = '1') or
                    (To_X01Z(Output_X_YADR) = '1') or
                    (To_X01Z(Output_X_SE) = '1')) then
                    for i in DOUT_zd'range loop
                        DOUT_zd(i) := 'X';
                    end loop;
                else
                    DOUT_zd := MemData(To_Integer(Address));
                end if;
            end if;
            VitalPathDelay01Z (
                OutSignal           => DOUT,
                GlitchData          => DOUT_GlitchData,
                OutSignalName       => "DOUT",
                OutTemp                 => DOUT_zd,
                Paths               => (0 => (XE_ipd'last_event, ZeroDelay, true),
                                            1 => (YE_ipd'last_event, ZeroDelay,
    true),
                                            2 => (OE_ipd'last_event, ZeroDelay,
    true),
                                            3 => (SE_ipd'last_event, ZeroDelay,
    true),
                                            4 => (XADR_ipd'last_event, ZeroDelay,
    true),
                                            5 => (YADR_ipd'last_event, ZeroDelay,
    true)),
                Mode                => OnDetect,
                Xon                     => Xon,
                MsgOn               => MsgOn,
                MsgSeverity         => WARNING);
    end if;
    end if;
    end process;


    -------------------------------------------------------------
    -- Assign address and check for unknown address or address change
    -- during program cycle.
    -------------------------------------------------------------
    --P2 : process(XE_ipd, XADR_ipd)
    --begin
    --    if (To_X01Z(CE_ipd) = '1') then
    --    if (NOW /= 0 ns) then
    --        if (To_X01Z(XE_ipd) = 'X') then
    --            Write_Msg("XE IS UNKNOWN.", Error);
```

```
--          end if;
--          if (To_Integer(XADR_ipd) = -1) then
--                Write_Msg("X ADDRESS IS UNKNOWN.", Error);
--          end if;
--          if (To_X01Z(Prog_Enable) = '1') then
--                Write_Msg("X ADDRESS CHANGED DURING PROGRAM CYCLE. ",
Error);
--          end if;
--          if (To_X01Z(Erase_Enable) = '1') then
--                Write_Msg("X ADDRESS CHANGED DURING ERASE CYCLE. ", Error);
--          end if;
--    end if;
--    end if;
--end process;
P3 : process(Prog_Enable, YE_ipd)           -- NISUNG
      variable PosTimeYE            : time := 0 ns;
      variable NegTimeYE            : time := 0 ns;
      variable OutLine        : line;
begin
    if (To_X01Z(Prog_Enable) = '1') then
            P3Violation <= '0';           -- NISUNG
            if ((YE_ipd'event) and (YE_ipd = '0')) then
                  NegTimeYE := NOW;
            end if;
            if ((YE_ipd'event) and (YE_ipd = '1')) then
                  PosTimeYE := NOW;
            end if;
            if ((NegTimeYE - PosTimeYE) > Tprogmax) then
                  Write(OutLine, string'("Timing Violation for Tprogmax"));
                  Write_Msg(OutLine, Error);
                  deallocate(OutLine);
                  P3Violation <= '1';       -- NISUNG
            end if;
      end if;
end process;
end BEHAVIORAL_SFB0008_08B9_CORE_MAIN_VITAL;
```

[0054] The invention claimed is: